An Adaptable Mobile Transaction Model

Patricia Serrano-Alvarado, Claudia Roncancio, Michel Adiba, Cyril Labbé

An Adaptable Mobile Transaction Model

Patricia Serrano-Alvarado, Claudia Roncancio, Michel Adiba, Cyril Labbé
e-mail: Firstname.Lastname@imag.fr

LSR-IMAG Laboratory BP 72, 38402 St. Martin d'Hères, France

Abstract

Mobile environments are characterized by high variability (e.g. variable bandwidth, disconnections, different communication prices) as well as by limited mobile host resources. Such characteristics lead to high rates of transaction failures and variable execution costs. To raise the success rate of transactions and to have a minimal control on resources consumption we claim that both application design and transaction management should be *environment aware*. This paper proposes an Adaptable Mobile Transaction model (AMT) that allows defining transactions with several *execution alternatives* associated to a particular context. When an AMT is launched, the appropriate execution alternative is initiated depending on the current environment state. The goal is to *adapt* transaction execution to context variations. Our model relaxes atomicity and isolation properties but preserves conflict-serializability. A specification of the AMT model in ACTA (formalism based on the first order logic) is presented. An analytical study shows that using AMTs increases commit probabilities and that it is possible to choose the way transactions will be executed according to their costs.

1 Introduction

The omnipresence of mobile devices such as cell phones, PDAs, smartcards, sensors and laptops, together with the development of different kinds of networks (local, wireless, ad-hoc, etc.) lead to a true mutation in the use, design and development of future information systems. Our work is related to the topics of ubiquitous and pervasive computing where technological improvements allow users to access data and perform transactions from any type of terminal and from anywhere using a wired or a wireless network. Applications that we have in mind cover a wide area. They might be personal ones, where clients want to access public data (e.g. weather forecast, stock exchange, road traffic) or professional ones, where mobility is inherent (e.g. mobile vendors/clients, health services, mobile offices, transport). We consider that mobile and fixed hosts can be clients or servers.

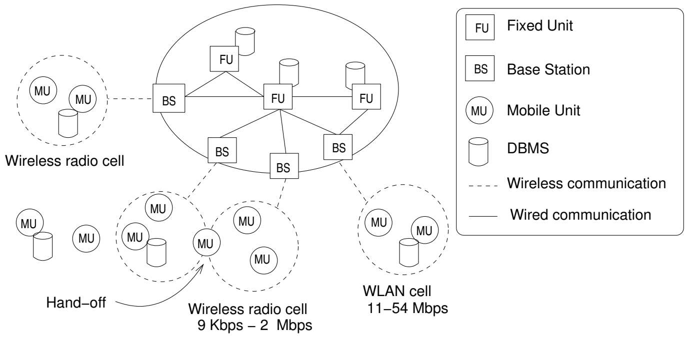

Figure 1: Mobile environment global architecture.

We consider a mobile computing environment with a network consisting of fixed and mobile hosts (FH, MH), see Figure 1. MHs could be of different nature ranging from PDAs to personal computers. Shared data are distributed over several database servers running, generally, on FHs. MH may run database management system (DBMS) modules and may provide some services to other hosts. While in motion, an MH may retain its network connection through a wireless interface supported by some FHs which act as Base Stations (BS). The geographical area covered by a BS is a cell. Each MH communicates with the BS covering its current cell. The process during which an MH enters into a new cell is called *hand-off*.

We make no specific assumptions about the database model (relational, object) but we place ourselves in a multidatabase environment assuming that data are managed by autonomous and possible heterogeneous DBMSs.

Applications in mobile environments are confronted to particular characteristics and limitations imposed by hardware such as: low and variable bandwidth, frequent disconnections, high communication prices, variable hardware configuration (due to plug-in components), limited display, battery autonomy, processing power and data storage. These limitations/variations lead to a lot of potential failure modes that affect data management process (e.g. queries, replication, caching, transactions, etc.).

In this paper, we are particularly interested on mobile transaction management. Focusing on this notion, we adopt a quite general definition: *a mobile transaction is a transaction where at least one mobile host takes part in its execution.* Mobile transactions are considered as long lived ones because of the probability of disconnections.

As an example, let us introduce an e-shopping application that allows people to browse products in an e-mall, to select, to book and to buy items. We assume also that secured e-payment is available based on credit cards or *e-money*. Application execution – as a set of transactions – will not be the same if they are launched from a (fixed) terminal office, from a PDA while traveling in a train or from home using a laptop. Thus, under this context:

- transactions may succeed but with different execution times (bandwidth capacity is highly variable) and communication prices (prices vary among wireless network providers/technologies/time-access);
- energy consumption is affected by low bandwidth (more battery is consumed);
- failures may occur due to unexpected disconnections or battery breakdown.

In traditional environments, application designers do not care about host and network characteristics. Nevertheless, in mobile applications we claim the necessity of being *environment aware* to overcome the infrastructure variability and to react to variations satisfying application and user requirements.

Several works concerning mobile transactions have been introduced (e.g. [10, 27, 9, 19, 27, 20, 26, 15, 18, 16, 6]). In the analysis made in [24], we found out that the majority of these proposals are particular solutions oriented to specific application contexts. Moreover, most of these works do not take into account the importance of mobile environment variability and therefore environment awareness.

The contribution of this paper is an Adaptable Mobile Transaction (AMT) model. The general idea is to define mobile transactions (T_{AMT}) with several *execution alternatives* associated to a particular mobile environment state. When a T_{AMT} transaction is launched, the appropriate execution alternative is initiated depending on the current mobile environment. We argue that adapting transaction executions improves commit rate, execution costs, response times and application availability, in short, the application's quality of service. We propose a formal specification of the AMT model using ACTA [7, 8]. The goal is to specify the properties and behaviour of the AMT model with a well defined and accepted formalism. Finally, we make an analytical study that shows how the AMT model increases transaction commit probability according to expected costs.

This paper is organized as follows: Section 2 presents the adaptable mobile transaction model and Section 3 its formal specification. Section 4 gives an analytical study of its performances. Section 5 discusses related work and Section 6 concludes this article.

2 Adaptability for Mobile Transactions

2.1 Overview

Traditionally, transactions are defined independently of execution infrastructure. This approach is well suited for centralized and distributed systems where the execution characteristics have acceptable, predictable and controlled state variations. As the mobile environment is highly variable, transaction execution can be unpredictable.

We consider that in order to optimize resources, both transaction design and management should be made taking into account environment awareness. Our proposal is done along these lines. For each mobile transaction, application programmers give execution alternatives suitable to a particular environment context. For instance, a transaction distributed over a fixed and a mobile host will be launched if a good connection is available, whereas, a local execution will be preferable if there is no connection or if only a very low bandwidth is available.

To allow context aware transaction executions we propose an Adaptable Mobile Transaction (AMT) model [23, 25]. This model offers concepts to design mobile transactions (T_{AMT}). Generally speaking, a T_{AMT} is composed of at least one *execution alternative* involving one or several mobile or fixed hosts. Execution alternatives may be semantically equivalent. The successful execution of one of them, represents a correct execution of the T_{AMT}. A T_{AMT} also contains *environment descriptors* which express the state of the mobile environment required to execute each alternative. When a T_{AMT} is launched, the mobile environment state is checked and the appropriate execution alternative is chosen – only one alternative must be active by T_{AMT}. If the environment state does not allow the execution of any alternative, the execution of the T_{AMT} may be deferred. As soon as an acceptable environment state will be detected, an execution alternative will be triggered.

2.2 Mobile Environment Awareness

Mobile environments include the wireless network (WN), MHs involved in mobile transactions and MH locations. Several dimensions – connection state, bandwidth-rate or communication price – characterize the state of a mobile environment. As said before, the variability of such environment may affect

	Dimension	States	Unit
WN	connection-state	*connected, disconnected*	
	bandwidth-rate	*high, medium, low*	kbytes/s
	communication-price	*free, cheap, expensive*	Euros/time
MH	available-battery	*full, half, low*	hh:mm:ss
	available-cache	*full, half, low*	kbytes
	available-persistent-memory	*full, half, low*	kbytes
	processing-capacity	*high, medium, low*	mhz/s
	estimated-connection-time	*t*	hh:mm:ss
	location		

Table 1: Mobile environment characteristics.

transaction execution and impact resources consumption. Environment descriptors introduced here reflect the execution context and its potential state variations. Thus, transaction designers who know the application characteristics and requirements for quality of service, specify different execution alternatives and the required execution context for each of them.[1]

The set of relevant dimensions is specific to the application environment. It depends on the mobile network, on the nature of MHs or on user behavior. For instance, network bandwith is important under packet-switched networks (e.g. UMTS) and not under circuit-switched ones (e.g. GSM) where bandwidth is guaranteed if the connection is available. Also, communication price is not relevant under WLAN networks but may be crucial in other environments. In addition, user-defined dimensions can be introduced, for instance, specific "quality" of data.

The basic set of dimensions we consider is introduced in Table 1. To simplify, *States* are divided in levels of quality – *high, medium, low* – nevertheless, they can be defined according to each specific dimension.

Environment descriptors (ED) indicate dimensions and states as follows.

Definition 1 *An Environment Descriptor* $ED = \{$*dimension=state(s)*$\}$ *contains a set of dimensions with their respective states at a given instant.*

For instance, to execute an alternative involving large data transfer, the required environment state may be:

Example 1 ED = {connection-state = *connected*, bandwidth-rate = *high*, communication-price= *free*, *cheap*}.

Notice that several hosts may be involved in a transaction. In that case, ED may or may not specify the required state for each involved host.

2.3 The AMT Model

This model allows to describe mobile transactions having one or more *execution alternatives* (EA_k), each of them is associated to an ED_k. EA_ks may take the form of any of the following execution types: the mobile transaction (1) is initiated by an MH and entirely executed on FHs, (2) is initiated by an MH/FH and entirely executed on an MH, (3) execution is distributed among MHs and FHs, and (4) execution is distributed among several MHs. So, a wide variety of mobile transactions is addressed.

An EA_k contains a set of *component transactions* (t_{ki}) which must respect ACID properties. They can be traditional *flat*, *distributed* or *close nested* transactions. *Compensating transactions* may be associated to component transactions. They will be executed in case of failures. An EA_k may be aborted if a component transaction aborts or if the mobile environment changes and the new state does not satisfy its environment descriptor. The EAs and the T_{AMT} are coordination units, data access is made only by component transactions. Making the analogy with multidatabases, component transactions are *local transactions* participating into *global transactions*.

Next, we present a semi-formal definition of the AMT model. A formal specification in ACTA [7, 8] is presented in Section 3.

Definition 2 *An adaptable mobile transaction is a* $T_{AMT} = \langle EA_k \rangle$ *where:*

- $\langle EA_k \rangle$, $k > 0$, *is a list of execution alternatives for* T_{AMT} *where* EA_k *has higher priority than* EA_{k+1}.
- $EA_k = (ED_k, EP_k)$, *an execution alternative has an execution plan* EP_k *to be executed if the actual mobile environment satisfies the environment descriptor* ED_k.
- ED_k *describes the environment state for the suitable execution of* EP_k.
- $EP_k = \{(t_{ki}, ct_{ki}, HostId)\}$, *is a set of triplets introducing a component transaction, its compensating one and the host where they have to be executed. Let* $\mathcal{RD}$ *be a relationship dependence over* EP_k, *such that:*
  $\forall (t_{ki}, ct_{ki}, HostId_x), (t_{kl}, ct_{kl}, HostId_y) \in EP_k;$
  $(t_{ki}, ct_{ki}, HostId_x) \mathcal{RD} (t_{kl}, ct_{kl}, HostId_y)$.

HostId indicates a database and the MH/FH responsible for the execution. Such host must execute only one component transaction per execution alternative. In $\mathcal{RD}$, we consider *parallel* or sequential execution dependencies.

Compensating transactions (ct_{ki}) are semantically equivalent to physical rollbacks and are defined to undo already committed component transactions. They recover semantically the database and avoid cascading aborts. Defining a ct_{ki} to compensate a t_{ki} is not always possible, thus, ct_{ki} will not always appear in EP_k.

[1]This might put the burden on the application designer. Future work should be oriented to develop computer aided environments for application developers.

AMT example

To continue with the example introduced in Section 1, consider an MH client with storage capacity, and an e-mall with two servers on the wired network: CatalogS and PurchaseS. The first site allows to query the store catalog whereas the second one takes purchase orders and payments.

We define the $T_{AMTshopping}$ with the following component transactions:

- GetCatalog allows clients to get a catalog;
- SelectItems allows clients to select items from a local copy of the catalog;
- Order-Pay allows clients to send the purchase order and payment to the store;
- AutoPay allows clients to pay on the MH (with e-money) without contacting other host;
- Order allows clients to send a purchase order (no payment included);
- Select-AutoPay = SelectItems + AutoPay

In Table 2, $T_{AMTshopping}$ proposes three execution alternatives to be triggered according to the environment state. In this example, wireless network dimensions that determine the choice of an alternative are: connection availability, bandwidth and communication price. The presence of the catalog on the MH is an application defined dimension. It takes the states *missing* (the catalog is not available on the MH) *present* (a version, probably out of date or incomplete, is on the MH) or *uptodate* (an up to date version is on the MH). Dimensions not appearing in environment descriptors are not considered as relevant for the context application.

In this example, the priorities between EA_ks are determined by the communication cost. Executing EA_1 is cheaper than executing EA_2. In EA_1 the MH has an up to date catalog, this allows saving communication messages. EA_1 may be launched even in disconnected mode and Order-Pay can be deferred until reconnection. EA_2 will be launched provided that the communication quality is acceptable (bandwidth-rate = *high*, *medium* and communication-price=*cheap*). EA_3 is executed even under bad communication rates (bandwidth-rate=*low*). The advantage of this alternative is that Select-AutoPay can be made in disconnected mode because the payment is in the MH. Order will be launched at reconnection.

Defining compensating transactions for this example is easy. They would mainly include operations to refund and cancel orders.

EA_k	ED_k	EP_k
k=1	{catalog-state= *uptodate*}	{(SelectItems, MH), (Order-Pay, PurchaseS)}
k=2	{connection-state=*connected*, bandwidth-rate = *high*, *medium*, communication-price=*cheap*, catalog-state=*present*, *missing*}	{(GetCatalog, CatalogS) (SelectItems, MH), (Order-Pay, PurchaseS)}
k=3	{connection-state=*connected*, bandwidth-rate=*low*, catalog-state=*missing*}	{(GetCatalog, CatalogS), (Select-AutoPay, MH), (Order, PurchaseS)}

Table 2: $T_{AMTshopping}$ example.

2.4 AMT Properties

A T_{AMT} can be considered at three different levels: the T_{AMT} itself, the execution alternatives and the component transactions. For the last ones we assume that ACID properties are guaranteed, nevertheless, as we will see latter, durability is conditioned by the success of the corresponding execution alternative.

An execution alternative (actually the associated EP_k) is a kind of *sagas* [13] containing a set of transactions which execution may be distributed among mobile and fixed hosts. The $\mathcal{RD}$ defined inside the alternative describes the possibility of a parallel or sequential execution of component transactions. Integrity constraints can be defined and verified at t_{ki} level, under the responsibility of the underlying DBMS. Global data integrity constraints are not considered but value dependencies between t_{ki}s (of the same EA_k) are allowed.

Atomicity and isolation for EA_k

Considering the restrictions of mobile environments, the AMT model relaxes atomicity by adopting semantic atomicity [12] (as in sagas).

Definition 3 ***Semantic atomicity of an*** EA_k
Each EA_k ensures semantic atomicity if either:

1. *all t_{ki}s defined in EA_k commit if EA_k commit*
2. *all t_{ki}s defined in EA_k are compensated or rolled back if EA_k aborts.*

The goal is to avoid blocking participant hosts and to allow MH disconnections. This is obtained with *local commits* where *partial* results are shared before the EA_k commits. The durability of locally committed transactions is conditioned to the commit of the EA_k. In case of abortion, compensating transactions are used.

To address critical applications with non-compensatable transactions, resources are blocked. Thus, when transactions terminate, resources are retained until a global decision (EA_k commits/aborts) is made. Hence, compensating transactions are not needed. If no participant commit locally, compensating transactions will be unnecessary and traditional atomicity is obtained.

Since dependency values between t_{ki}s of the same EA_k are allowed, a correct global ordering must be

Property	t_{ki}	$\mathbf{EA}_k$	T_{AMT}
Atomicity	√	Semantic atomicity	Semi-atomicity
Consistency	√	Semantic consistency	
Isolation	√	Relaxed (local commits)	
Durability	√ conditioned	Underlying DBMS	
Correctness	Serializability	Global serializability	

Table 3: Summary of AMT properties.

ensured. That is because concurrent execution of several alternatives might introduce indirect interference between value dependent transactions. The criterion used to control the correctness of concurrent execution alternatives is *global serializability*.[2] Global serializability states that transactions of each EA_k must have the same relative serialization order in their corresponding underlying DBMS.

Even though a global serializable order is preserved, semantic consistency [12] is provided – due to semantic atomicity.

Atomicity and Isolation for T_{AMT}

We mentioned in Section 2.3 that EA_ks defined in a T_{AMT} may be semantically equivalent. A T_{AMT} is correctly executed if one of its EA_ks is successfully executed. This results in *semi-atomicity* [28] which is guaranteed for T_{AMT}s as follows.

Definition 4 ***Semi-atomicity of a*** T_{AMT}
Each T_{AMT} garantees semi-atomicity if either:

1. *the commit of a T_{AMT} implies the commit of only one EA_k and the abortion or compensation of all component transactions of other EA_l*
2. *the abortion of a T_{AMT} implies the abortion or compensation of all other component transactions of the EA_k in progress.*

Serializability of T_{AMT}s is offered trough the serializability of execution alternatives.

Definition 5 ***Global serializability of*** EA_k
A set of EA_k ensures global serializability:

1. *if the execution order of component transactions ensures a serializable order in each site and*
2. *if EA_k ensures also a serializable order on all sites.*

Regarding the durability property, once the corresponding EA_k commits (and consequently the T_{AMT}), durability of component transactions is provided by the underlying DBMS. Table 3 summarizes properties at all levels (t_{ki}, EA_k, T_{AMT}).

In [25, 23], we define a middleware (named TransMobi) that implements the AMT model with appropriate protocols. Due to space constraints it is not presented here. TransMobi uses a client/agent/server architecture. It manages environment awareness based on events that are generated thanks to sensors that supervise MH and wireless communication capacities. Roughly speaking, applications request AMT executions to TransMobi which verifies the mobile environment state and decides the way transactions will be executed (it chooses the appropriate execution alternative).

As a middleware between application code and existing DBMSs, TransMobi coordinates the execution of T_{AMT}s. We assume the existence of DBMS functionalities on fixed and mobile hosts. TransMobi relies on them for the execution – ensuring ACID properties – of component and compensating transactions. EA_k and T_{AMT} properties are ensured as follows. Concerning EA_k, semantic atomicity is guaranteed by the CO2PC protocol that we propose.[3] Semantic consistency is a consequence of the execution of compensating transactions. As the AMT model is an open nested transaction, isolation at EA_k and AMT levels is relaxed. To guarantee global serializability, we propose to use the Optimistic Ticket Method [14] that is used in multidatabase systems. Finally, as each T_{AMT} has only one EA_k active, the commit/abort of EA_k ensures the semi-atomicity of T_{AMT}.

3 AMT Formal Specification

This section proposes a specification of the AMT model in ACTA [7, 8]. ACTA is a formalism based on first order logic that allows defining and comparing principal characteristics of extended transaction models. With ACTA, it is possible to specify the effects of extended transactions on each other and on objects. We use ACTA because it is a well-accepted formalism and its capabilities of expression and extension are enough to specify the AMT properties.

This section is organized as follows. Firstly, an axiomatic definition of the AMT model (Section 3.1) is proposed. Secondly, in order to obtain the properties of the T_{AMT} transactions, we analyze existing axioms (Section 3.2).

3.1 AMT axiomatic definition

Dependencies introduced in ACTA are not sufficient to specify the AMT model. Nevertheless, due to the extensibility capability of the formalism, we propose the following dependency:

Unique Begin Dependency ($t_j \; \mathcal{UBD} \; t_i$): t_i can begin, if any other t_j has not begun:
$(begin_{t_i} \in H) \Rightarrow \neg(begin_{t_j} \in H)$

[2] In this paper, serializability is actually *conflict-serializability.*

[3] [3, 2] introduce a comparison of CO2PC with other validation protocols for mobile environments.

Notation
Next, we show the elements and sets used in the AMT axiomatic definition.

- T_{AMT} defines an *Adaptable Mobile Transaction* that contains a list of Execution Alternatives EA_k.

  $T_{AMT} = \langle EA_1, ..., EA_n \rangle,\ n > 0$

- An execution alternative EA_k is composed of a set of component transactions t_{ki}.

  $EA_k = \{t_{k1}, ..., t_{kn}\},\ n > 0$

  $EA_k \neq EA_l$

  Here, we make abstraction of environment descriptor ED_k and execution site $SiteId$ described in Definition 2.

- EA_k organize its t_{ki} in two sets, CT_k et NT_k, where CT_k contains *compensables* t_{ki} and NT_k contains *non-compensables* t_{ki}.

  $CT_k \cap NT_k = \phi$

  Compensables and non-compensables transaction sets must de disjoints.

- ST_k denotes the t_{ki} list that must be executed sequentially:

  $(ST_k \subseteq CT_k) \vee ((t_{ki}, ..., t_{kn-1} \subseteq CT_k) \wedge (t_{kn} \in NT_k))$

  where $1 \leq i \leq n, t_{ki} \in ST_k$

  Only compensable transactions can be executed sequentially, except for the last one.

- ct_{ki} denotes a *compensating transaction* for t_{ki}: $\forall t_{ki} \in CT_k\ \exists\ ct_{ki}$.

  All compensable transactions must have a compensating transaction.

- t denotes t_{ki} or ct_{ki}.

- S_u denotes the transaction set that is executed on site u. EA_k can execute only one component transaction by site.

Annex A.1 presents the types of dependencies used in the next axiomatic definition of the AMT model.

Definition 6 ***Axiomatic definition of the AMT model***

1. $SE_{T_{AMT}} = SE_{EA} = SE_t =$ {begin, commit, abort}

2. $IE_{T_{AMT}},\ IE_{EA},\ IE_t =$ {begin}

3. $TE_{T_{AMT}},\ TE_{EA},\ TE_t =$ {commit, abort}

4. t satisfies the fundamental axioms I to IV (c.f. Annex A.2)

5. $View_{T_{AMT}} = \phi$

6. $View_{EA} = \phi$

7. $View_t = H^{(S_u)}$

8. $ConflictSet_{T_{AMT}} = \phi$

9. $ConflictSet_{EA} = \phi$

10. $ConflictSet_t = \{p_{t'}[ob] \mid t' \neq t, t', t \in S_u,\ Inprogress(p_{t'}[ob])\}$

11. $\forall ob\ \exists p\ (p_t[ob] \in H) \Rightarrow (ob \text{ is atomic})$
    ob is *correct* and *serializable.*

12. $(commit_t \in H) \Rightarrow \neg(t\, \mathcal{C}^*\, t)$
    t can commit *locally* if it is not part of a cycle.

13. $\exists ob\ \exists p\ (commit_t[p_t[ob]] \in H) \Rightarrow (commit_t \in H)$
    if $p_t[ob]$ commits t must commit.

14. $(commit_t \in H) \Rightarrow \forall ob\ \forall p\ ((p_t[ob] \in H) \Rightarrow (commit_t[p_t[ob]] \in H))$
    If t commits all its operations must commit.

15. $\exists ob\ \exists p\ (abort_t[p_t[ob]] \in H) \Rightarrow (abort_t \in H)$
    If $p_t[ob]$ aborts t must abort.

16. $(abort_t \in H) \Rightarrow \forall ob\ \forall p\ ((p_t[ob] \in H) \Rightarrow (abort_t[p_t[ob]] \in H))$
    If t aborts all its operations must abort.

17. $(commit_{EA} \in H) \Rightarrow \neg(EA\, \mathcal{R}^*\, EA)$
    EA commits globally if it is not part of a cycle.

18. $post(begin_{T_{AMT}}) \Rightarrow (((begin_{EA_k} \in H) \Rightarrow ConditionEnvironment) \wedge ((EA_l\, \mathcal{UBD}\, EA_k) \in DepSet_{ct}) \wedge (t_{ki}\, \mathcal{BD}\, EA_k))$

    where $1 \leq k \leq m,\ 1 \leq l \leq m,\ k \neq l$

    EA_k begins if it satisfies the environment condition. Only one EA_k of a T_{AMT} must be initiated. A t_{ki} cannot begin if its EA_k has not begun.

19. $post(begin_{EA_k}) \Rightarrow (((T_{AMT}\mathcal{AD}EA_k) \in DepSet_{ct}) \wedge ((EA_k\, \mathcal{AD}\, T_{AMT}) \in DepSet_{ct}) \wedge (t_{ki} \in TS_k) \Rightarrow ((t_{ki}\, \mathcal{BCD}\, t_{ki-1}) \in DepSet_{ct}))$

    where $1 \leq k \leq m$,

    If EA_k aborts T_{AMT} must abort and if T_{AMT} aborts EA_k must abort. Component transactions of ST_k are executed sequentially, other t_{ki} can be executed concurrently.

20. $post(begin_{t_{ki}}) \Rightarrow (((EA_k\, \mathcal{AD}\, t_{ki}) \in DepSet_{ct}) \wedge ((t_{ki} \in CT_k) \Rightarrow ((t_{ki}\, \mathcal{WD}\, EA_k) \in DepSet_{ct}) \wedge ((ct_{ki}\, \mathcal{BCD}\, t_{ki}) \in DepSet_{ct})) \wedge ((t_{ki} \in NT_k) \Rightarrow ((t_{ki}\, \mathcal{AD}\, EA_k) \in DepSet_{ct})))$

    where $1 \leq i \leq n,\ 1 \leq k \leq m$

If t_{ki} aborts EA_k aborts. For compensable transactions, if EA_k aborts t_{ki} must abort (if t_{ki} has not committed) and ct_{ki} can begin only if t_{ki} has committed. For non-compensable transactions (NT_k), if EA_k aborts t_{ki} must abort. t_{ki} commit is delayed until EA_k commit (t_{ki} wait for an EA_k abort until EA_k commits).

21. $post(commit_{t_{ki}}) \Rightarrow (((ct_{ki}\mathcal{BAD}EA_k) \in DepSet_{ct}) \wedge ((ct_{ki}\,\mathcal{CMD}\,EA_k) \in DepSet_{ct}))$

    where $1 \leq i \leq n,\ 1 \leq k \leq m$

    ct_{ki} can begin if EA_k aborts and if EA_k aborts after t_{ki} commits, ct_{ki} must commit.

3.2 Deductions and analysis of the axioms

This section deduces the properties of T_{AMT} transactions from axioms of Definition 6.

Axioms 1-3 specify significant events of the AMT at three levels (T_{AMT}, EA and t). Axiom 4 states that component and compensating transactions (t) must respect fundamental axioms (c.f. Annex A.2). Axioms 5-6 show that T_{AMT} and EA_k are control points which do not access data. Thus, Axioms 8-9 specify that there are not possible conflicts for T_{AMT} and EA_k. Axiom 7 states that the view of t is limited to the projection of history H on S_u (set of transactions executing on site u). Therefore, in Axiom 10 the set of conflicts of t is composed of all operations executed by other transactions in S_u.

In Axiom 18, dependency $((begin_{EA_k} \in H) \Rightarrow ConditionEnvironment)$, introduces a beginning condition for EA_k. The goal is to launch the alternative when *ConditionEnvironment* is satisfied. *ConditionEnvironment* becomes true when the *environment descriptor* (ED) of EA_k (c.f. Section 2.3) matches with the current mobile environment state.

Next section (3.2.1), shows properties of component and compensating transactions. Sections 3.2.2 and 3.2.3 deduce semantic atomicity and global serializability of EA_k respectively. Finally, Section 3.2.4 shows the semi-atomicity property of T_{AMT}.

3.2.1 t_{ki} and ct_{ki} properties

This section shows that component and compensating transactions are atomic (Lemma 2) so with AID properties.

Lemma 1 ***If*** $t_i \in T_{AMT}$, t_i ***is failure atomic***

Proof of Lemma 1:

t_i is *failure atomic* if it satisfies both conditions of Definition 2 (c.f. Annex A.3).

1. Condition 1 (all clause) is derived from Axioms 13 and 14.
2. Condition 2 (nothing clause) is derived from Axioms 15 and 16.

Lemma 2 ***If*** $t_i \in T_{AMT}$, t_i ***is an atomic transaction***

Proof of Lemma 2 :

t_i is an atomic transaction if it satisfies both conditions of Theorem 1

1. Condition 1 (*failure atomic*) is derived from Lemma 1.
2. Condition 2 (serializable) is derived from Axioms 11 and 12.

Since atomic transactions satisfy AID properties (c.f. Theorem 1), component and compensating transactions are AID transactions.

3.2.2 Semantic atomicity of EA_k

This section shows that execution alternatives have the semantic atomicity property.[4] We begin by introducing the commitment (Lemma 3) and abortion (Lemma 4) of EA_k. Next, we obtain the semantic atomicity of EA_k (Lemma 5).

Lemma 3 ***Commitment of an*** EA_k
Let H be the history of an execution alternative EA_k with n component transactions.
$((commit_{EA_k} \in H) \Rightarrow \forall i, 1 \leq i \leq n\ (commit_{t_{ki}} \in H))$

The commitment of an execution alternative EA_k implies the commitment of all associated component transactions t_{ki}.

Proof of Lemma 3:

If EA_k commits, its set of component transactions must commit due to the abort dependency of EA_k on t_{ki} of the Axiom 20 (the first one) and the fundamental Axiom III:
$\forall i, 1 \leq i \leq n\ ((abort_{t_{ki}} \in H) \Rightarrow (abort_{EA_k} \in H)) \Leftrightarrow ((commit_{EA_k} \in H) \Rightarrow (commit_{t_{ki}} \in H))$

Lemma 4 ***Abortion of an*** EA_k
Let H be a history of an EA_k with n component transactions.
$(abort_{EA_k} \in H) \Rightarrow \forall i, 1 \leq i \leq n,\ \forall j, 1 \leq j \leq n, i \neq j\ ((abort_{t_{ki}} \in H) \wedge (commit_{t_{kj}} \rightarrow commit_{ct_{kj}}))$

If EA_k aborts, associated component transactions must be aborted or compensated.

[4] Semantic atomicity expression is similar to the one introduced in [7].

Proof of Lemma 4:

Case 1. If EA_k aborts when t_{ki} is in progress, t_{ki} aborts due to the $\mathcal{WD}$ or $\mathcal{AD}$ dependencies of t_{ki} on EA_k (Axiom 20). Due to fundamental Axiom II, it is not necessary to specify that only transactions that have begun are aborted. Similarly, due to fundamental Axiom III, it is not necessary to specify that only non committed transactions are aborted. Due to the $\mathcal{BCD}$ dependency of ct_{ki} on t_{ki} (Axiom 20) ct_{ki} does not begin in this case:

$(abort_{EA_k}) \Rightarrow \forall i, 1 \leq i \leq n\ (abort_{t_{ki}} \in H)$

Only $t_{ki} \in CT_k$ can commit due to the $\mathcal{WD}$ dependency. By its side, $t_{ki} \in NT_k$ commit until EA_k commit due to the abortion dependence of t_{ki} on EA_k. Thus, all $t_{ki} \in NT_k$ in progress are aborted if EA_k aborts.

Case 2.

1. If EA_k aborts after t_{ki} commits and before a t_{kj} begins, ct_{ki} must commit due to the $\mathcal{CMD}$ dependency of Axiom 21:
   $(abort_{EA_k} \in H) \Rightarrow \forall i, 1 \leq i \leq m\ (commit_{ct_{ki}})$
2. Due to the existence of $\mathcal{BCD}$ dependency of ct_{ki} on t_{ki} on axiom 20, if ct_{ki} begins, t_{ki} has been committed:
   $(begin_{ct_{ki}} \in H) \Rightarrow (commit_{t_{ki}} \rightarrow begin_{ct_{ki}})$
   By fundamental Axiom II:
   $(commit_{ct_{ki}} \in H) \Rightarrow (begin_{ct_{ki}} \rightarrow commit_{ct_{ki}})$
   Thus, by the semantics of the dependency relation the commit of ct_{ki} is done after the commit of t_{ki}:
   $(commit_{ct_{ki}} \in H) \Rightarrow (commit_{t_{ki}} \rightarrow commit_{ct_{ki}})$
3. From 1 and 2:
   $((abort_{EA_k} \in H) \Rightarrow \forall i, 1 \leq i \leq m\ (commit_{ct_{ki}})) \wedge$
   $((commit_{ct_{ki}} \in H) \Rightarrow (commit_{t_{ki}} \rightarrow commit_{ct_{ki}}))$
   Simplifying:
   $(abort_{EA_k} \in H) \Rightarrow \forall i, 1 \leq i \leq n(commit_{t_{ki}} \rightarrow commit_{ct_{ki}})$

Case 3. If an EA_k aborts when a t_{ki} is in progress (Case 1) and after a t_{kj} has committed (Case 2):

$(abort_{EA_k} \in H) \Rightarrow \forall i, 1 \leq i \leq n,\ \forall j, 1 \leq j \leq n, i \neq j$
$((abort_{t_{ki}} \in H) \wedge (commit_{t_{kj}} \rightarrow commit_{ct_{kj}}))$

Lemma 5 ***Each*** EA_k ***ensures semantic atomicity***

Each EA_k ensures semantic atomicity if conditions of definition 3 are ensured. [5]

[5] [23] presents a specification in ACTA of Definitions 3,4 and 5.

Proof of Lemma 5:

1. Condition 1 of Definition 3 (if EA_k commits, all t_{ki} transactions must commit) is ensured by Lemma 3.
2. Condition 2 of Definition 3 (if EA_k aborts, all t_{ki} transactions must abort or compensate) is ensured by Lemma 4.

3.2.3 Global serializability of EA_k

This section specifies that execution alternatives are globally serializable. From some axioms and Definition 6, it is possible to deduce the global serializability of EA_k (Lemma 6).

Lemma 6 EA_k ***ensures global serializability***

Proof of Lemma 6:

To provide global serializability both conditions of Definition 5 must be ensured:

1. Condition 1 is derived from Axiom 12.
2. Condition 2 is derived from Axiom 17.

3.2.4 Semi-atomicity of T_{AMT}

This section states that T_{AMT} have the semi-atomicity property. We begin by introducing the commitment of T_{AMT} (Lemmas 7 and 8) as well as its abortion (Lemmas 9 and 10). Next, we obtain the semi-atomicity of T_{AMT} (Lemma 11).

To obtain those properties, we define firstly the commitment (complete) of a T_{AMT} (Lemma 8). For this, we specify the commitment (simple) of a T_{AMT} (Lemma 7) and the commitment of an EA_k (Lemma 3). Similarly, we define the abortion of a T_{AMT} (Lemmas 10, 9 and 4).

Lemma 7 ***Commitment of a*** T_{AMT}
Let H be the history of a T_{AMT} and EA_k an execution alternative associated to a T_{AMT}.
$((commit_{T_{AMT}} \in H) \Rightarrow \exists k, 1 \leq k \leq m\ (commit_{EA_k} \in H))$

This Lemma expresses that if the history contains the commit of a T_{AMT} it contains also the commit of an associated execution alternative k.

Proof of Lemma 7:

1. If T_{AMT} commits, EA_k must also commit due to the abort dependency of T_{AMT} on EA_k specified in Axiom 19 (the first one) and the fundamental Axiom III, which says that a transaction must commit or abort:

   $\forall k, 1 \leq k \leq m\ ((abort_{EA_k} \in H) \Rightarrow (abort_{T_{AMT}} \in H)) \Leftrightarrow ((commit_{T_{AMT}} \in H) \Rightarrow (commit_{EA_k} \in H))$

2. Only one EA_k commits due to the $\mathcal{UBD}$ dependency of the Axiom 18 where only one EA must begin:
   $\forall k,\ 1 \leq k \leq m,\ \forall l,\ 1 \leq l \leq m,\ l \neq k$
   $(begin_{EA_k} \in H) \Rightarrow \neg(begin_{EA_l} \in H)$

3. From 1 and 2
   $((commit_{T_{AMT}} \in H) \Rightarrow \exists k, 1 \leq k \leq m\ (commit_{EA_k} \in H))$

Lemma 8 ***Complete commitment of a T_{AMT}***
Let H be the history of a T_{AMT} with n component transactions and EA_k an execution alternative associated to T_{AMT}.
$((commit_{T_{AMT}} \in H) \Rightarrow \exists k, 1 \leq k \leq m\ (commit_{EA_k} \in H)) \wedge$
$((commit_{EA_k} \in H) \Rightarrow \forall i, 1 \leq i \leq n\ (commit_{t_{ki}} \in H))$

Simplifying:
$(commit_{T_{AMT}} \in H) \Rightarrow \exists k, 1 \leq k \leq m\ \forall i, 1 \leq i \leq n,\ (commit_{t_{ki}} \in H)$

Proof of Lemma 8:

This Lemma follows from Lemmas 7 et 3.

Lemma 9 ***Abortion of a T_{AMT}***
Let H be the history of a T_{AMT} and EA_k an execution alternative associated to T_{AMT}.
$((abort_{T_{AMT}} \in H) \Rightarrow \exists k, 1 \leq k \leq m\ ((abort_{EA_k} \in H))$

This Lemma expresses the history in which the abortion of a T_{AMT} implies the abortion of the EA_k in progress.

Proof of Lemma 9:

1. If T_{AMT} aborts, EA_k must also abort due to the abort dependency of EA_k on T_{AMT} of Axiom 19:
   $(abort_{T_{AMT}} \in H) \Rightarrow (abort_{EA_k} \in H)$

2. Only one EA_k aborts due to the $\mathcal{UBD}$ dependency of the Axiom 18 where only one EA_k must begin, see 2 in Lemma 7

Lemma 10 ***Complete abortion of a T_{AMT}***
Let H be the history of a T_{AMT} with n component transactions and EA_k an execution alternative associated to T_{AMT}.
$((abort_{T_{AMT}} \in H) \Rightarrow \exists k, 1 \leq k \leq m((abort_{EA_k} \in H)) \wedge$
$(abort_{EA_k} \in H) \Rightarrow \forall i, 1 \leq i \leq n,\ \forall j, 1 \leq j \leq n, i \neq j$
$((abort_{t_{ki}} \in H) \wedge (commit_{t_{kj}} \rightarrow commit_{ct_{kj}}))$

Simplifying:
$(abort_{T_{AMT}} \in H) \Rightarrow \forall i, 1 \leq i \leq n,\ \forall j, 1 \leq j \leq n, i \neq j$
$((abort_{t_{ki}} \in H) \wedge (commit_{t_{kj}} \rightarrow commit_{ct_{kj}}))$
This Lemma expresses the history in which the abortion of a T_{AMT} implies the abortion of the EA_k in progress and the compensation or abortion of component transactions associated to EA_k.

Proof of Lemma 10:

This Lemma follows from Lemmas 9 and 4.

Thanks to the analysis made, we can state that:

Theorem 1 ***The execution of a T_{AMT} produces one of the following histories:***

1. $((commit_{T_{AMT}} \in H) \Rightarrow \exists k, 1 \leq k \leq m\ (commit_{EA_k} \in H)) \wedge$
   $((commit_{EA_k} \in H) \Rightarrow \forall i, 1 \leq i \leq n\ (commit_{t_{ki}} \in H))$

2. $((abort_{T_{AMT}} \in H) \Rightarrow \exists k, 1 \leq k \leq m((abort_{EA_k} \in H)) \wedge$
   $(abort_{EA_k} \in H) \Rightarrow \forall i, 1 \leq i \leq n,\ \forall j, 1 \leq j \leq n, i \neq j$
   $((abort_{t_{ki}} \in H) \wedge (commit_{t_{kj}} \rightarrow commit_{ct_{kj}}))$

Proof of Theorem 1:

This Theorem follows from Lemmas 8 and 10.

Lemma 11 ***Each T_{AMT} guarantee the semi-atomicity***
Each T_{AMT} guarantees semi-atomicity if both conditions of Definition 4 are satisfied.

Proof of Lemma 11:

1. Condition 1 of Definition 4 (all transactions in EA_k commit) is satisfied by Lemma 8.

   Abortion or/and compensation of all component transactions of other EA_l is not necessary due to $\mathcal{UBD}$ dependence of Axiom 18.

2. Condition 2 of Definition 4 (all transactions in T_{AMT} are aborted or compensated) is satisfied by Lemma 10.

Theorem 2 ***Each T_{AMT} has the following properties:***

(1) t_i has the AID properties,

(2) EA_k ensures semantic atomicity,

(3) EA_k ensures global serializability,

(4) T_{AMT} ensures semi-atomicity.

Proof of Theorem 2

(1) follows from Lemma 2, (2) follows from Lemma 5, (3) follows from Lemma 6 and (4) follows from Lemma 11.

Definition 7 *The management schema of a T_{AMT} is correct if it follows the Definition 6.*

4 Impact of Environment Awareness: Analytical Study

This section provides an analytical study of the capabilities allowed by the AMT model. In Section 4.1, we use a probabilistic model to analyze several execution alternatives one by one – separated of the AMT model. For each of them, we evaluate its initiation probability and its execution cost. Section 4.2 highlights the benefits expected from environment awareness and AMT adaptable facilities. It is shown that the T_{AMT} initiation probability is always greater than the initiation probability of one alternative. In addition, it is shown how the AMT model allows the designer to define the best T_{AMT} according to the required quality of service: low cost without complete guarantee of success or at the opposite success any time at any cost. The $T_{AMTshopping}$ example (introduced in Section 2.3) is used all along this section to illustrate the analytical study. A short concluding discussion is proposed in Section 4.3.

4.1 Analytical Study of an Execution Alternative

Mobile Environment Model

As mentioned in previous sections, a mobile environment state can be seen as a set of dimensions. An EA_k is initiated only if the environment state satisfies the acceptable states for each dimension.

Example 2 *The environment descriptors used by $T_{AMTshopping}$ include the following dimensions and states:*

		$j = 1$ Good	$j = 2$ Medium	$j = 3$ Bad
$i = 1$	connection-state	*connected*		*disconnected*
$i = 2$	bandwidth-rate	*high*	*medium*	*low*
$i = 3$	communication-price		*cheap*	*expensive*
$i = 4$	catalog-state	*uptodate*	*present*	*missing*

Definition 8 *Let p_{ij} be the probability of dimension i to be in state j.*

In the resulting matrix $P = (p_{ij})$, $\forall i$, $\sum_j p_{ij} = 1$. Next example shows that this matrix depends on the considered mobile environment.

Example 3 *An example of the matrix probability of an environment like the one introduced in the example 2 could be :*

$$P = (p_{ij}) = \begin{bmatrix} 0.8 & 0 & 0.2 \\ 0.7 & 0.2 & 0.1 \\ 0 & 0.4 & 0.6 \\ 0.2 & 0.3 & 0.5 \end{bmatrix}$$

Here, the probability of being *connected* is given by p_{11}, the probability of having an *uptodate* catalog is given by p_{41} and so on. p_{31} has 0 probability because in the considered environment, communication is never *free*.

Environment Description Matrix

Definition 9 *For each EA_k=(ED_k, EP_k) we denote by Δ^k the boolean matrix where:*

$$\delta^k_{ij} = \begin{cases} 1 & if\ the\ state\ j\ of\ dimension\ i\ is \\ & acceptable\ for\ EA_k \\ 0 & otherwise \end{cases}$$

Example 4 *The Δ^k matrix for the three proposed alternatives of $T_{AMTshopping}$ (see Table 2) are:*

$$\Delta^1 = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix} \Delta^2 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \Delta^3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

If a dimension i does not appear in ED_k, any state is suitable: if $i \notin ED_k$ then $\forall j, \delta^k_{ij} = 1$. Δ^1 and Δ^3 illustrate this case.

Cost Matrix

Definition 10 *Let C^k be the matrix where c^k_{ij} is the cost of the EP_k execution in state j of dimension i.*

c^k_{ij} must be defined by the designer according to application needs in cost improvement. Example 5 shows some particular definitions of cost.

Example 5 *For the $T_{AMTshopping}$ one could identify the memory utilization as a cost associated to the dimension* catalog-state *or the CPU consumption as a cost associated to the dimension* connection-state *(since in disconnected mode more operations are done on the MH). Let us focus on communication price and battery utilization respectively associated to dimension* communication-price *and* bandwidth-rate *(bandwidth limitations increase battery consumption). The considered wireless network is UMTS which uses a packet-switched communication where bandwidth rate goes from 144 kbps (vehicular mobility), 384 kbps (pedestrian mobility) to 2 mbps for indoor traffic. These bandwidth rates correspond to high, medium and low states. Communication price depends on the size of transmitted data (packets). For instance, the execution of EP_1 requires three wireless logical messages. First, a transaction request (kind of login), then the purchase order (component transaction* Order-Pay*) and finally, a message is received by the MH to confirm the order (acknowledgment). The execution of EP_2 and EP_3 requires an extra message to ask for the catalog which is received through another message (*GetCatalog*).*

Three different sizes of messages may be identified: small messages (login, ack and ask for catalogue), medium ones (for Order-Pay *and* Order*) and large ones (for* GetCatalog*). Let us assume that small, medium and large messages are composed respectively of* 1, 10 *and* 20 *packets. So, if the execution plan of EA_k sends n_s small, n_m medium and n_l large messages then n_p*

is the number of packets sent or received[6] *by the MH during the* EP_k *execution,* $n_p = n_s + 10n_m + 20n_l$.

To evaluate communication cost (communication price and battery consumption), we assume that:

- *Sending a single packet in the state* cheap *(resp. expensive) costs 1 unit of price (resp. 2 units).*
- *If bandwidth is high (resp. medium, low) sending/receiving a single packet uses* 0.1% *(resp.* 0.2%, 0.4%*) of the battery capacity. In this case, the cost associated to the dimension* bandwidth-rate *is the battery consumption.*

Under these assumptions we have:

$$C^k = \begin{bmatrix} 0 & 0 & 0 \\ 0.1n_p & 0.2n_p & 0.4n_p \\ 0 & n_p & 2n_p \\ 0 & 0 & 0 \end{bmatrix}$$

where c^k_{2j} *is the battery consumption of the* EP_k *execution when the* bandwidth-rate *is in state* j. c^k_{3j} *is the price of the* EP_k *execution when the* communication-cost *is in state* j. *In* $T_{AMTshopping}$, $n_p = 12$ *for* EP_1 *and* $n_p = 33$ *for* EP_2 *and* EP_3*, so:*

$$C^1 = \begin{bmatrix} 0 & 0 & 0 \\ 1.2 & 2.4 & 4.8 \\ 0 & 12 & 24 \\ 0 & 0 & 0 \end{bmatrix} \quad C^2 = C^3 = \begin{bmatrix} 0 & 0 & 0 \\ 3.3 & 6.6 & 13.2 \\ 0 & 33 & 66 \\ 0 & 0 & 0 \end{bmatrix}$$

Mean Cost of an EA_k

The execution plan of EA_k is launched by the system when the mobile environment is in the state j of the dimension i with the probability:

$$\frac{\delta^k_{ij} p_{ij}}{\sum_j \delta^k_{ij} p_{ij}}$$

So, the mean cost due to dimension i of the EP_k execution is given by:

$$c^k_i = \frac{\sum_j \delta^k_{ij} c^k_{ij} p_{ij}}{\sum_j \delta^k_{ij} p_{ij}}$$

Example 6 *With previous* P, Δ^k *and* C^k *we can compute:*

$$c^3_2 = \frac{13.2 * 0.1}{0.1} = 13.2\%$$

$$c^3_3 = \frac{33 * 0.4 + 66 * 0.6}{0.4 + 0.6} = 52.8$$

This means that the average of battery consumption of EA_3 is 13.2% with an average price of $c^3_3 = 52.8$ units, whereas $c^1_2 = 1.8\%$, $c^1_3 = 19.2$ units and $c^2_2 = 4.03\%$, $c^2_3 = 33$ units.

[6] Eventually, it could be interesting to distinguish between sending or receiving messages.

EA_k Initiation Probability

Definition 11 *Let* q^k *be the probability for* EA_k *of being selected for execution.* q^k *will be called the* EA_k *initiation probability. Since the probability that dimension* i *has an acceptable state for* EA_k *is given by* $\sum_j \delta^k_{ij} p_{ij}$*, we have:*

$$q^k = \prod_i (\sum_j \delta^k_{ij} p_{ij})$$

So the execution plan of an EA_k has a chance to be initiated ($q^k > 0$) iff $\forall i, \exists j$ such that $\delta^k_{ij} = 1$.

Comparing EAs

Example 7 *In order to study EAs in different types of environment, performances indices are studied in regard to the probability of the bandwidth to be low* (p_{23})*:*

$$P = \begin{bmatrix} 0.8 & 0 & 0.2 \\ (1 - p_{23})/2 & (1 - p_{23})/2 & p_{23} \\ 0 & 0.4 & 0.6 \\ 0.2 & 0.3 & 0.5 \end{bmatrix}$$

Fig. 2 shows the EA_k initiation probability and Fig. 3 the battery consumption associated to each EA_k in $T_{AMTshopping}$. It can be seen that if the bandwidth is often *low* (e.g. $p_{23} = 0.8$) then EA_3 is the alternative with the highest battery consumption (see Fig. 3) whereas it has the best initiation probability (see Fig. 2). q^1 is constant because it does not depends on the bandwidth to be *low* but only on the probability for the catalog to be *uptodate*.

4.2 Analytical Study of a T_{AMT}

As mentioned in Section 2, mobile environment awareness allows the system to choose an EA_k if the environment state corresponds to the associated ED_k. EA_k has higher priority than EA_{k+1}; this allows us to assume without lost of generality that:

Property 1 *If a state of the environment is suitable for an* EA_k *it should not be suitable for an* $EA_{k'}$ *of the same* T_{AMT}*. That is:* $\forall (k, k'), k \neq k', \exists i$ *such that* $\forall j, \delta^{k'}_{ij} \neq \delta^k_{ij}$

T_{AMT} Initiation Probability

Definition 12 *Let* q_{AMT} *be the* T_{AMT} *initiation probability. Thanks to property 1, we have:* $q_{AMT} = \sum_k q^k$

Mean Cost of a T_{AMT}

Definition 13 *Let* c_i *be the mean cost (associated to dimension i) of the execution of* T_{AMT}*. Under the assumption that the environment is stable during the execution,* EA_k *is initiated with the probability* q^k *so the cost of the whole* T_{AMT} *is* c^k_i *with the probability* q^k*, hence:*

$$c_i = \frac{\sum_k c^k_i q^k}{\sum_k q^k}$$

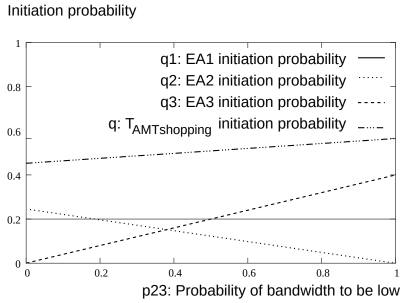

Figure 2: Initiation probability vs probability of bandwidth to be *low*.

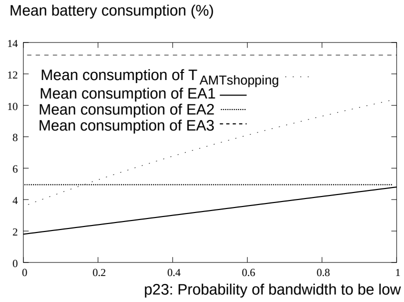

Figure 3: Battery consumption vs probability of bandwidth to be *low*.

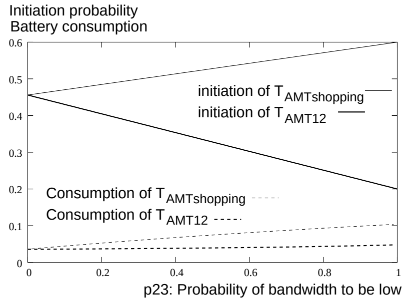

Figure 4: Initiation probability and battery consumption vs bandwidth.

As an example, we study the mean battery consumption and the initiation probability of $T_{AMTshopping}$ in regards to the probability of the bandwidth to be *low* (p_{23}).

Fig. 2 shows that the $T_{AMTshopping}$ initiation probability never reaches 1. This is due to the fact that for certain states no alternative is defined. For instance, if the MH is *disconnected* with a *missing* catalog $T_{AMTshopping}$ can not be initiated. This figure also shows that the initiation probability of a single EA_k is smaller than the initiation probability of the whole $T_{AMTshopping}$.

Fig. 3 shows that the mean battery consumption of the $T_{AMTshopping}$ increases when the probability of the bandwidth to be *low* is going up. This is due to the fact that EA_3 initiation probability is bigger than the one of EA_2 (see Fig. 2).

Fig. 4 shows performance parameters for $T_{AMTshopping}$ and a variant without EA_3. This new transaction is called T_{AMT12}. It can be seen that $T_{AMTshopping}$ has a better initiation probability whereas T_{AMT12} has a better mean battery consumption.

4.3 Discussion

This section showed that compared to non-adaptable approaches, adaptability in transaction execution improves performances and allows choosing the way the transaction will be executed according to execution costs. Without environment awareness, a transaction is defined in a standard way (the execution plan is fixed). The system will try to execute this transaction whatever the state of the environment is. If the current state does not allow the execution, the transaction will fail even if another execution alternative could have been successful. In the same way, the environment state may lead to a costly execution without considering cheaper alternatives. Allowing the system to choose the execution plan in regards to the current environment state is a way to ensure better performances.

With n different execution plans, environment awareness and AMT facilities allow to choose among $n!$ different T_{AMT} (each one including n alternatives) depending upon the user optimization criteria, e.g. minimal costs or execution time. The definition of the T_{AMT} that provides best initiation probability does not depends on the number of defined EA_ks but on the capacity to overcome environment variations.

To ensure a better quality of service with T_{AMT} execution, specific tools based on this analytical model could be used to define optimized T_{AMT}s.

Environment awareness allows to define the T_{AMT} that fits the best to the quality of service required by the application, for instance, a trade-off between reducing costs and relaxing quality of service can be done.

5 Related Work

The AMT model was inspired from DOM [4] and Flex [11] where the general idea is to define *equivalent* transactions for being executed in case of failures. The DOM transaction model allows *close* and *open nested* transactions. Compensating transactions as well as contingency ones can be specified for being executed if a given transaction fails. In the Flex transaction model, contingency transactions are defined in terms of *functionally equivalent* transactions. A failure order is defined where the execution of a transaction depends on the failure of another one. Unlike AMT, DOM and Flex transactions are not defined to deal with mobile environments and the notion of context awareness is not considered.

The panorama of mobile transactions is vast. A detailed analysis of several works is given in [24]. In general, the adaptability vision is very limited, almost all studied systems adapt their behavior to support disconnections. That is the case in Clustering [21], Two-tier replication [15], HiCoMo [18], IOT [22], Pro-motion [26], Prewrite[20], MDSTPM[27] and Pre-serialization [9]. Only Moflex [16] addresses execution adaptability when hand-off occur. On the contrary, our proposition – AMT model – allows adaptation to any defined environment characteristic.

Concerning execution types, several works consider transaction initiated by an MH and completely executed on FHs. For instance, KT [10], MDSTPM, Moflex and Pre-serialization. Other proposals focus on transaction execution on an MH: HiCoMo, IOT and Pro-motion. Only clustering and Two-tier replication consider two execution types, (1) on an MH (during disconnections) and (2) distributed among an MH and FHs (during connections). None of analyzed proposals face distributed executions among several MHs or among mobile and fixed hosts. TCOT [17] and UCM [1] consider those kind of distributed executions even though these works address principally the transaction validation process. With the AMT model it is possible to define transactions following the five execution types mentioned here. Indeed, supporting different execution types facilitates AMT adaptation to mobile environments.

Although data replication is not necessarily a transactional issue, it is at the heart of several works on mobile transactions. Propositions like Clustering, Two-tier replication and IOT consider that MHs contain replicated data. During disconnections, copies are modified trough a kind of second class transactions. At reconnection, different reconciliation process are proposed (re-executions with first class transactions, synchronization of copies, etc.). Our work does not address replication aspects, we separate transaction and replication issues. We consider a multidatabases system where each site (mobile or fixe) is independent of each other. A similar approach is taken in KT [10], Pre-serialization, MDSTPM and Moflex. Nevertheless, in these works the multidatabases environment is composed only of DBMSs installed on FHs. MHs are considered only to request transactions.

Finally, the principal difference of the AMT model and existent works is that the description of the environment execution is attached to the transaction definition.

6 Conclusion and Perspectives

This paper addressed mobile transactions and made several contributions: (1) We proposed the AMT model inspired by previous extended transaction models. However, we put a special emphasis on environment awareness by considering specific dimensions, e.g. bandwidth rate, connection state, mobile host resources, etc. The model concerns transactions involving several heterogeneous DBMS running on mobile or fixed hosts. (2) We introduced a formal specification of the proposed model, this will allow to compare our proposal with other advanced transaction models. (3) We provided an analytical study that can be viewed as a semi automatic tool to optimize mobile transaction design and executions. This is done by an a priori study of several execution alternatives and their respective probabilities of success.

Research perspectives include: (1) The implementation of an application developer utility in order to facilitate the design of AMT transactions. Environment awareness can be used to profile the application environment and users habits. Once the environment profiled, an analytical calculus – like the one presented here – could be done to dynamically optimize T_{AMT} in regards to required quality of service. (2) The analysis of the suitability of adapting transactions not only at the beginning (as done in our current proposal) but also during their execution; aspects related to reusing work already done by the adapted transaction have to be explored. (3) Applying the AMT model to peer-to-peer and network ad-hoc environments; as the mobile environment they are characterized by high variability.

References

[1] C. Bobineau. *Gestion de transactions en environnement mobile*. PhD thesis, Université de Versailles, Versailles, France, 2002. In French.

[2] C. Bobineau, C. Labé, C. Roncancio, and P. Serrano-Alvarado. Comparing Transaction Commit Protocols for Mobile Environments. In *Int. DEXA Workshop on Mobility in Databases and Distributed Systems (MDDS)*, Zaragoza, Spain, September 2004.

[3] C. Bobineau, C. Labé, C. Roncancio, and P. Serrano-Alvarado. Performances de Protocoles Transactionnels en Environnement Mobile. In *20émes Journées Bases de Données Avancées (BDA)*, Montpellier, France, October 2004. In French.

[4] A. Buchmann, M. T. Özsu, M. Hornick, D. Georgakopoulos, and F. A. Manola. *Database Transaction Models for Advanced Applications*, chapter 5, A Transaction Model for Active Distributed Object Systems. Morgan Kaufmann Publisher, 1992.

[5] P. K. Chrysanthis. *ACTA, A Framework for Modeling and Reasoning about Extended Transactions*. PhD thesis, University of Massachusetts, Amherst, USA, 1991.

[6] P. K. Chrysanthis. Transaction Processing in a Mobile Computing Environment. In *IEEE Workshop on Advances in Parallel and Distributed Systems (APADS)*, Princeton, USA, October 1993.

[7] P. K. Chrysanthis and K. Ramamritham. *Database Transaction Models for Advanced Applications*, chapter 10, ACTA: The SAGA Continues. Morgan Kaufmann Publisher, 1992.

[8] P. K. Chrysanthis and K. Ramamritham. Synthesis of Extended Transaction Models Using ACTA. *ACM Transactions on Database Systems (TODS)*, 19(3), 1994.

[9] R. A. Dirckze and L. Gruenwald. A Pre-Serialization Transaction Management Technique for Mobile Multidatabases. *Mobile Networks and Applications (MONET)*, 5(4), 2000.

[10] M. H. Dunham, A. Helal, and S. Balakrishnan. A Mobile Transaction Model that Captures Both the Data and the Movement Behavior. *ACM/Baltzer Journal on special topics in mobile networks and applications*, 2(2), 1997.

[11] A. K. Elmagarmid, Y. Leu, and M. Rusinkiewics. A Multidatabase Transaction Model for INTERBASE. In *Int. Conf. on Very Large Databases (VLDB)*, Brisbane, Australia, August 1990.

[12] H. García-Molina. Using Semantic Knowledge for Transaction Processing in a Distributed Database.

ACM Transactions on Database Systems (TODS), 8(2), 1983.

[13] H. García-Molina and K. Salem. Sagas. In *ACM SIGMOD Conference*, San Francisco, USA, May 1987.

[14] D. Georgakopoulos, M. Rusinkiewicz, and A. Sheth. Using Tickets to Enforce the Serializability of Multidatabase Transactions. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 6(1), 1994.

[15] J. Gray, P. Helland, P.O'Neil, and D. Shasha. The Dangers of Replication and a Solution. In *ACM SIGMOD Conference*, Montreal, Canada, June 1996.

[16] K. Ku and Y. Kim. Moflex Transaction Model for Mobile Heterogeneous Multidatabase Systems. In *IEEE Workshop on Research Issues in Data Engineering*, San Diego, USA, February 2000.

[17] V. Kumar, N. Prabhu, M. H. Dunham, and A. Y. Seydim. TCOT- A Timeout-Based Mobile Transaction Commitment Protocol. *IEEE Transactions on Computers*, 51(10), 2002.

[18] M. Lee and S. Helal. HiCoMo: High Commit Mobile Transactions. *Kluwer Academic Publishers Distributed and Parallel Databases (DAPD)*, 11(1), 2002.

[19] Q. Lu and M. Satynarayanan. Improving Data Consistency in Mobile Computing Using Isolation-Only Transactions. In *IEEE HotOS Topics Workshop*, Orcas Island, USA, May 1995.

[20] S. K. Madria and B. Bhargava. A Transaction Model for Improving Data Availability in Mobile Computing. *Kluwer Academic Publishers Distributed and Parallel Databases (DAPD)*, 10(2), 2001.

[21] E. Pitoura and B. Bhargava. Data Consistency in Intermittently Connected Distributed Systems. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 11(6), 1999.

[22] M. Satynarayanan, J. Kistler, P. Kumar, E. Okasaki, H. Siegel, and C. Steere. Coda: A Highly Available File System for Distributed Workstation Environment. *IEEE Transactions on Computers*, 39(4), 1990.

[23] P. Serrano-Alvarado. *Transactions Adaptables pour les Environnements Mobiles*. PhD thesis, Université Joseph Fourier, Grenoble, France, 2004. In French.

[24] P. Serrano-Alvarado, C. Roncancio, and M. Adiba. A Survey of Mobile Transactions. *Kluwer Academic Publishers Distributed and Parallel Databases (DAPD)*, 16(2), 2004.

[25] P. Serrano-Alvarado, C. Roncancio, M. Adiba, and C. Labbé. Environment Awareness in Adaptable Mobile Transactions. In *Journées de Bases de Données Avancées (BDA)*, Lyon, France, October 2003.

[26] G. D. Walborn and P. K. Chrysanthis. Transaction Processing in PRO-MOTION. In *ACM Symp. on Applied Computing*, San Antonio, USA, February 1999.

[27] L. H. Yeo and A. Zaslavsky. Submission of Transactions from Mobile Workstations in a Cooperative Multidatabase Processing Environment. In *Int. Conf. on Distributed Computing Systems (ICDCS)*, Poznan, Poland, June 1994.

[28] A. Zhang, M. Nodine, B. Bhargava, and O. Bukhres. Ensuring Relaxed Atomicity for Flexible Transactions in Multidatabase Systems. In *ACM SIGMOD Conference*, Minneapolis, USA, May 1994.

Annex A

A.1 Types of dependencies used in the AMT specification

Here there is a review of the types of dependencies used in axioms of the AMT model.

Begin Dependency ($t_j\ \mathcal{BD}\ t_i$): t_j cannot begin executing until t_i has begun:
$(begin_{t_j} \in H) \Rightarrow (begin_{t_i} \rightarrow begin_{t_j})$.

Abort Dependency ($t_j\ \mathcal{AD}\ t_i$): if t_i aborts then t_j aborts:
$(abort_{t_i} \in H) \Rightarrow (abort_{t_j} \in H)$.

Begin-on-Commit Dependency ($t_j\ \mathcal{BCD}\ t_i$): t_j cannot begin executing until t_i commits:
$(begin_{t_j} \in H) \Rightarrow (commit_{t_i} \rightarrow begin_{t_j})$.

Weak-Abort Dependency ($t_j\ \mathcal{WD}\ t_i$): if t_i aborts and t_j has not committed then t_j must abort:
$(abort_{t_i} \in H) \Rightarrow (\neg(commit_{t_j} \rightarrow abort_{t_i}) \Rightarrow (abort_{t_j} \in H))$.

Begin-on-Abort Dependency ($t_j\ \mathcal{BAD}\ t_i$): t_j cannot begin executing until t_i aborts:
$(begin_{t_j} \in H) \Rightarrow (abort_{t_i} \rightarrow begin_{t_j})$.

Force-Commit-on-Abort Dependency ($t_j\ \mathcal{CMD}\ t_i$): if t_i aborts, t_j must commit:
$(abort_{t_i} \in H) \Rightarrow (commit_{t_j} \in H)$.

A.2 Fundamental axioms

Definition 1 ***Fundamental axioms of transactions***

I $\forall \alpha \in EI_t\ (\alpha \in H^t) \Rightarrow \beta \in EI_t\ (\alpha \rightarrow \beta)$
A transaction cannot be initieted by two different events.

II $\forall \delta \in ET_t\ \exists \alpha \in EI_t\ (\delta \in H^t) \Rightarrow (\alpha \rightarrow \delta)$
If a transaction has terminated, it must have been preiously initiated.

III $\forall \gamma \in ET_t\ (\gamma \in H^t) \Rightarrow \delta \in ET_t\ (\gamma \rightarrow \delta)$
A transaction cannot be terminated by two different events.

IV $\forall ob\ \forall p\ (p_t[ob] \in H) \Rightarrow ((\exists \alpha \in EI_t(\alpha \rightarrow p_t[ob])) \wedge (\exists \gamma \in ET_t(p_t[ob] \rightarrow \gamma)))$
Only in progress transactions can invoke operations on objects.

A.3 Failure atomicity

Definition 2 ***Failure atomicity***

A transaction t is failure atomic if :

1. $\exists ob\ \exists p\ (commit[p_t[ob]] \in H) \Rightarrow$
$\forall ob'\ \forall q\ ((q_t[ob'] \in H) \Rightarrow (commit[q_t[ob'] \in H))$
2. $\exists ob\ \exists p\ (abort[p_t[ob]] \in H) \Rightarrow$
$\forall ob'\ \forall q\ ((q_t[ob'] \in H) \Rightarrow (abort[p_t[ob']] \in H))$

Theorem 1 ***Properties of atomic transactions:***

1. If t is an atomic transaction, t is **failure atomic**,
2. A set T of committed atomic transactions is **serializable**.

Proof of Theorem 1

The proof is donne in [5].